

Encoding problems as SMT

Solver interaction, correctness, and alternative encodings

CMPUT 664 • Formal Verification

Decidability and completeness ask different questions

Fix a theory T and a sentence F .

Decidability asks for one algorithm that works for every input sentence:

- it always terminates, and
- it returns the correct answer.

Completeness of T asks whether the theory settles every sentence:

$$T \models F \quad \text{or} \quad T \models \neg F.$$

Addition and multiplication over integers

Presburger arithmetic allows quantifiers, addition, and order over integers. It is decidable.

Allowing multiplication between unknown integers adds equations such as

$$p(x_1, \dots, x_n) = 0.$$

The solver question is therefore

$$\exists x_1, \dots, x_n \in \mathbb{Z}. \ p(x_1, \dots, x_n) = 0.$$

Many individual equations are easy. However, no algorithm can decide every integer polynomial equation and always terminate. ([Matiyasevich's theorem](#))

PA is more expressible than simply integer equations

For an integer polynomial equation, the question is

Do any integer values satisfy $p(x_1, \dots, x_n) = 0$?

For Peano arithmetic, the question is

Can the axioms of PA prove the sentence F ?

where F can include natural numbers, allows arbitrary quantifiers, and includes induction.

Why integers and real closed fields behave differently

Recall that solving for polynomials in the domain of **real closed fields** is decidable.

Both domains are infinite, and both theories use addition and multiplication. However, their structures give those symbols different expressive power.

Over $\mathbb{N}$ or $\mathbb{Z}$, the discrete positions

$$0, 1, 2, \dots$$

let arithmetic encode an arbitrarily long finite computation as one number.

In a real closed field, no formula defines exactly those discrete positions. Quantifier elimination instead reduces a formula to finitely many polynomial sign conditions.

The rational field $\mathbb{Q}$ is a separate case. Its full first-order theory is undecidable. Whether every polynomial equation over $\mathbb{Q}$ can be decided remains open.

Today

1. Solver interface and organization
2. Combining theory reasoning
3. Modeling choices
4. Criteria for evaluating an SMT encoding
5. Graph coloring as a case study
6. Scheduling as a second case study

Using an SMT solver

Interface, outcomes, and organization

A basic solver query

```
1 (declare-const x Int)      // Need to define the sort of the constants
2 (declare-const y Int)
3
4 (assert (> x 0))
5 (assert (= y (+ x 1)))
6 (assert (< y 4))
7
8 (check-sat)
9 (get-model)
```

Possible solver outcomes

Result	Meaning
sat	A model satisfies all assertions
unsat	No model satisfies all assertions
unknown	The solver did not establish either result

For a decidable fragment, a decision procedure exists in principle. A particular solver may lack complete support or reach a resource limit.

For an undecidable fragment, no algorithm can terminate with the correct answer on every input. Solvers can still answer many individual inputs.

Read a model

For the previous query, one possible model is

```
1 (define-fun x () Int 1)
2 (define-fun y () Int 2)
```

Models are not generally unique. The solver may return any assignment satisfying every assertion.

Boolean and theory reasoning

An SMT formula has two related levels:

1. A Boolean structure over theory atoms
2. Theory constraints represented by those atoms

For

$$(x < y \vee z = 0) \wedge (y \leq x),$$

the SAT layer selects a Boolean combination; theory reasoning checks whether the selected arithmetic constraints can hold together.

This general organization is commonly called **DPLL(T)**.

A theory conflict

Suppose the Boolean search selects

$$x < y \quad \text{and} \quad y \leq x.$$

The Boolean assignment alone is consistent: both atoms were assigned true.

Linear arithmetic determines that the two constraints cannot both hold. The solver rejects that Boolean case and continues.

Modeling choices

The selected theory is part of the model

Identify the required semantics

Before writing constraints, decide (for instance):

- Which values are Boolean, integer, real, or finite-width?
- Is overflow possible and relevant?
- Are collections maps, sequences, sets, or finite tuples?
- Which operations need exact semantics?
- Are quantifiers necessary?

Classify each constraint

Which theory or theories are needed?

1. $0 \leq i < n$

2. $x \& 15 = 0$

3. $a[i + 1] = a[i] + 1$

4. $f(x) = y \wedge x = z$

5. s begins with "GET "

Some constraints require a combination of theories.

From a problem to a formula

An input instance I belongs to some source problem.

An encoding constructs an SMT formula:

$$\text{encode}(I) = F_I .$$

The central correctness condition

For a decision problem, we usually require

$$I \text{ has a solution} \iff F_I \text{ is satisfiable.}$$

This statement relates two different semantic domains: source solutions and SMT models.

Two directions must be justified

Soundness of decoding

$$M \models F_I \quad \Rightarrow \quad \text{decode}(M) \text{ solves } I.$$

Completeness of encoding

$$S \text{ solves } I \quad \Rightarrow \quad \exists M. M \models F_I \wedge \text{decode}(M) = S.$$

Criteria for an encoding

An encoding should be evaluated along several dimensions:

Criterion	Question
Correctness	Do SMT models correspond to source solutions?
Representation	Do variables and theories match the source semantics?
Efficiency	What formula does the solver receive, and how does it behave?
Diagnosis	Can models and inconsistent constraints be interpreted?

Case study: graph coloring

Comparing two representations of one problem

The source problem

Given an undirected graph $G = (V, E)$ and a number of colors k , **assign each vertex one color so that adjacent vertices receive different colors.**

For every edge $(u, v) \in E$:

$$\text{color}(u) \neq \text{color}(v).$$

The decision question is whether such an assignment exists.

A concrete graph

Let

$$V = \{a, b, c, d\}$$

and

$$E = \{(a, b), (a, c), (b, c), (b, d), (c, d)\}.$$

Vertices a, b, c form one triangle; vertices b, c, d form another.

Two encodings

Integer representation

One integer c_v per vertex

The value of c_v is the color

Domain constraints restrict values

Boolean one-hot representation

One Boolean $x_{v,i}$ per vertex-color pair

$x_{v,i}$ states that v uses color i

Exactly-one constraints connect choices

Integer encoding: variables and domains

Introduce one integer variable for each vertex:

$$c_a, c_b, c_c, c_d \in \mathbb{Z}.$$

For each $v \in V$, constrain

$$0 \leq c_v < k.$$

The integer c_v represents the color assigned to v .

Integer encoding: edges

For every $(u, v) \in E$, add

$$c_u \neq c_v.$$

For the concrete graph:

$$c_a \neq c_b \wedge c_a \neq c_c \wedge c_b \neq c_c \wedge c_b \neq c_d \wedge c_c \neq c_d.$$

A model for three colors

One satisfying model is

$$c_a = 2, \quad c_b = 0, \quad c_c = 1, \quad c_d = 2.$$

Decoding reads each value as the corresponding vertex color.

Boolean encoding: variables

Introduce a Boolean variable for every vertex-color pair:

$$x_{v,i} \quad v \in V, 0 \leq i < k.$$

$x_{v,i}$ means that vertex v is assigned color i .

This representation has $|V| * k$ variables.

Each vertex has exactly one color

At least one color:

$$\bigvee_{0 \leq i < k} x_{v,i}.$$

At most one color:

$$\bigwedge_{0 \leq i < j < k} (\neg x_{v,i} \vee \neg x_{v,j}).$$

Both parts are required for each vertex v .

Adjacent vertices use different colors

For every edge $(u, v) \in E$ and color i , add

$$\neg x_{u,i} \vee \neg x_{v,i}.$$

Equivalently:

$$\neg(x_{u,i} \wedge x_{v,i}).$$

Correctness of the integer encoding

From a model to a coloring:

- Domain constraints assign each vertex a color in $\{0, \dots, k - 1\}$.
- Edge constraints give adjacent vertices different values.

From a coloring to a model, assign $M(c_v) = C(v)$ for every vertex. The coloring establishes all domain and edge constraints.

Correctness of the Boolean encoding

From a model to a coloring:

- Exactly-one constraints select one color for each vertex.
- Edge-color constraints prevent adjacent vertices from selecting the same color.

From a coloring to a model, set exactly the corresponding $x_{v,i}$ variables to true.

Two colors or three?

For the concrete graph G , consider

$$F_{G,2} \quad \text{and} \quad F_{G,3} .$$

$F_{G,2}$ is unsatisfiable because the graph contains a triangle. $F_{G,3}$ is satisfiable.

Unsatisfiable cores

Suppose a formula is a conjunction of labeled assertions:

$$F = C_1 \wedge \cdots \wedge C_n.$$

An **unsatisfiable core** is a subset of these assertions whose conjunction is already unsatisfiable.

A core need not be minimum or even minimal.

A core for two-coloring

The constraints involving only a, b, c already encode a two-coloring of a triangle:

- Domains for c_a, c_b, c_c
- $c_a \neq c_b$
- $c_a \neq c_c$
- $c_b \neq c_c$

They form an unsatisfiable subset. Constraints involving d are unnecessary for this explanation.

Comparing the two encodings

Let $n = |V|$, $m = |E|$, and use k colors.

Encoding	Variables	Main constraints
Integer colors	n integers	n domains, m disequalities
Boolean one-hot	nk Booleans	n at-least-one, $n\binom{k}{2}$ at-most-one, mk edge constraints

Formula size alone does not determine solver performance.

What encoding efficiency includes

Relevant costs include:

- Time and memory required to construct the formula
- Numbers of variables, assertions, and auxiliary terms
- Strength of deductions exposed by the representation
- Solver time and memory across representative instances
- Cost of decoding and validating models

Structural tradeoffs in the case study

The integer encoding is compact and delegates finite-domain reasoning to arithmetic.

The Boolean encoding is larger, but each possible color is explicit and available to Boolean propagation.

Alternative exactly-one encodings change both size and propagation behavior.

There is no encoding that is uniformly best for every graph family and solver.

Single-machine scheduling

Arithmetic and disjunctive constraints

The source problem

Three jobs have fixed durations:

Job	<i>A</i>	<i>B</i>	<i>C</i>
Duration	2	3	2

Requirements:

- Jobs execute on one machine and may not overlap.
- *A* must finish before *C* starts.
- Every job must finish by time 7.

Start-time variables

Introduce integer start times

$$s_A, s_B, s_C \in \mathbb{Z}.$$

Nonnegative start times and the deadline give

$$0 \leq s_J \quad \text{and} \quad s_J + d_J \leq 7$$

for each job J with duration d_J .

Precedence

“ A finishes before C starts” becomes

$$s_A + 2 \leq s_C.$$

Using $s_A < s_C$ would be insufficient: it compares start times but ignores the duration of A .

Non-overlap

For every pair of distinct jobs J, K , one must finish before the other starts:

$$s_J + d_J \leq s_K \quad \vee \quad s_K + d_K \leq s_J .$$

This is a disjunction of linear arithmetic constraints.

A satisfying schedule

One model is

$$s_A = 0, \quad s_C = 2, \quad s_B = 4.$$

The occupied intervals are

$$A : [0, 2), \quad C : [2, 4), \quad B : [4, 7).$$

Omitting non-overlap changes the problem

Without the pairwise non-overlap constraints, a model may place jobs on top of each other:

$$s_A = 0, \quad s_B = 0, \quad s_C = 2.$$

This model satisfies the domains, deadline, and precedence constraint—but it is not a valid single-machine schedule.

Correctness argument for scheduling

From an SMT model to a schedule:

- Start-time and deadline constraints give valid intervals.
- Pairwise disjunctions establish non-overlap.
- The precedence constraint establishes the required order.

From a valid schedule to a model, assign each s_J its scheduled start time. Every encoded constraint then follows from the source requirements.

Time-indexed encodings

The source problem is **non-preemptive**: once a job starts, it executes continuously for d_J time units.

For integer time slots $t \in \{0, \dots, 6\}$, two Boolean encodings are possible:

- A **start indicator** records that job J starts at t . Exactly one start and the duration determine one continuous block.
- An **execution indicator** records that job J executes during slot t . Duration, machine capacity, and contiguity constraints preserve the original problem.

Without contiguity constraints, execution indicators instead encode **preemptive scheduling**: a job may execute, pause while another job runs, and resume later.

Contiguity for execution indicators

For every $i < j < k$, require

$$(y_{J,i} \wedge y_{J,k}) \Rightarrow y_{J,j} ,$$

or equivalently

$$\neg y_{J,i} \vee y_{J,j} \vee \neg y_{J,k} .$$

If a job executes in two slots, it must execute in every slot between them. Together with the duration constraint, this gives one continuous block.

Evaluating an encoding

Correctness, efficiency, and diagnosis

Sorts determine possible models

Replacing integer start times by real start times admits fractional schedules.

Replacing mathematical integers by bit-vectors introduces finite ranges and overflow.

Using Booleans for a finite choice requires explicit constraints connecting those Booleans.

The selected sort must match the source domain.

Generated constraints for finite instances

For a finite graph, edge constraints can be generated explicitly:

$$\bigwedge_{(u,v) \in E} c_u \neq c_v.$$

This is different from asking the solver to reason about a quantified graph theory.

The host program constructs a finite, quantifier-free formula. This often gives more predictable solver behavior, at the cost of formula size that grows with the instance.

Validate decoded models

For small instances, independently check that a decoded model satisfies the source problem:

- Every value belongs to its intended domain.
- Every source constraint holds.
- Auxiliary variables are decoded consistently.

Validation can expose missing constraints and incorrect decoding even when the solver is correct.

Encoding review questions

1. What does each variable represent?
2. How is each source requirement encoded?
3. How is an SMT model decoded?
4. Why is every decoded model valid?
5. Can every source solution be represented?
6. What is the expected formula size and structure?
7. Which symmetries or redundant constraints affect solving?
8. How can satisfiable and unsatisfiable results be diagnosed?

Next time

Specifications and Hoare logic